Performance Tuning with Shark on Mac and iPhone

Session 413

These are confidential sessions—please refrain from streaming, blogging, or taking pictures.

At the convention center, a movie will be placed here

Performance Tuning with Shark on Mac and iPhone

Wade Tregaskis
Minion, Developer Technologies

Why Are You Here?

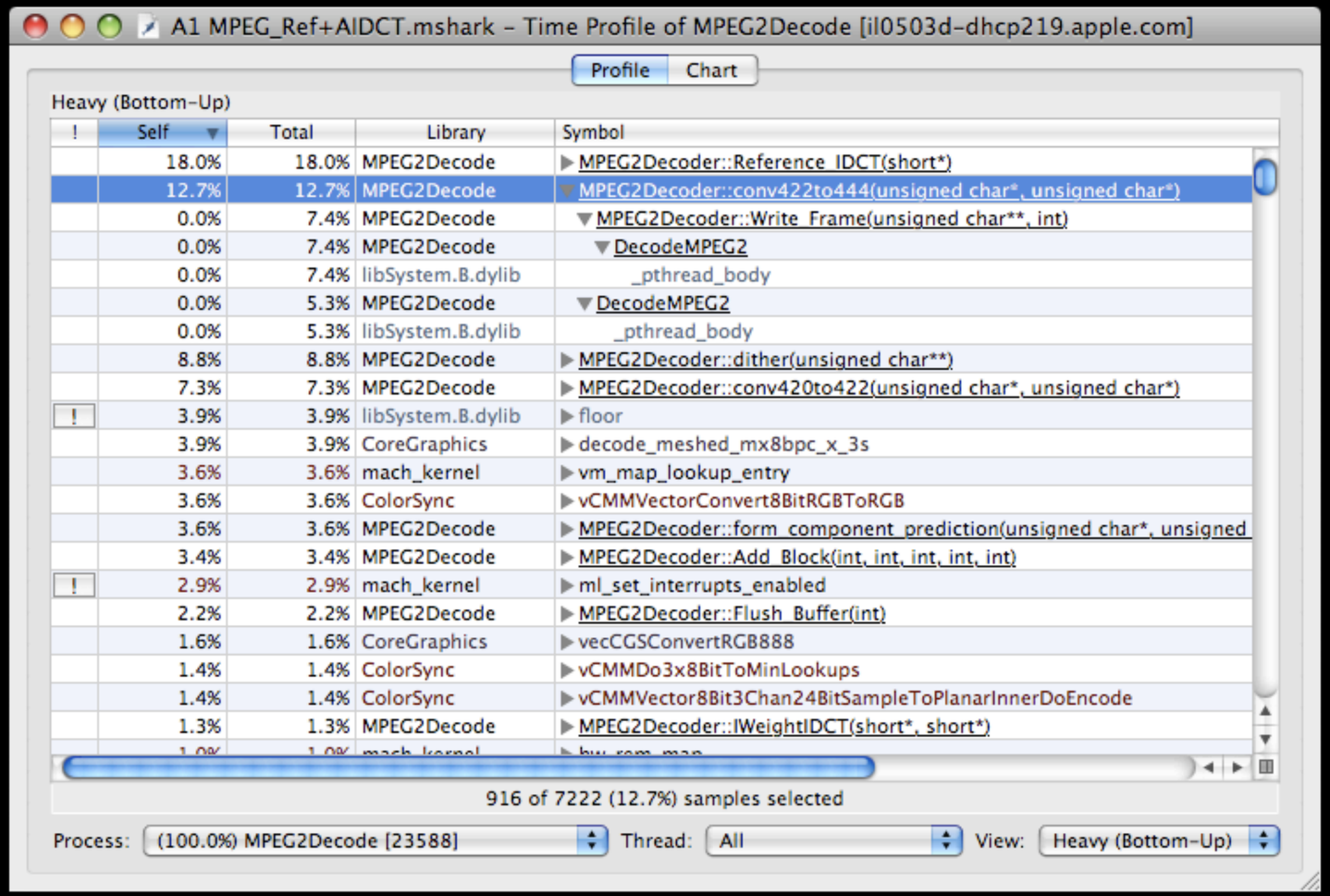
Technically speaking

- Speed
  - Get things done faster
  - Do more things
- Usability
  - Responsiveness
- Endurance
  - Longer battery life
- Efficiency
  - Meaningful work per watt

How Does Shark Help?

Time Profile

- Shows you where you're spending CPU time

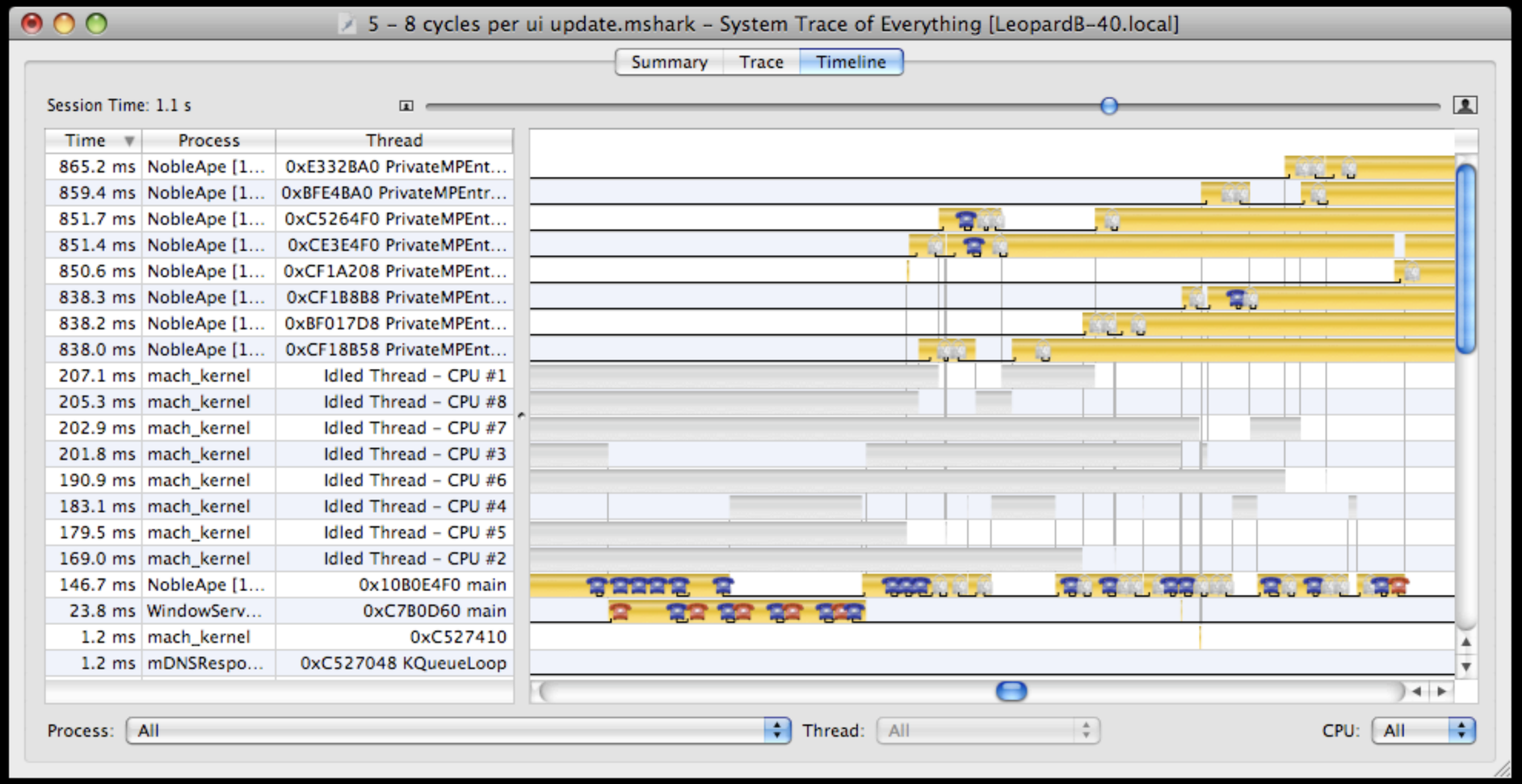


How Does Shark Help?

System Trace

Only on Mac OS

- Thread scheduling, system calls, Mach messages, VM faults, etc.

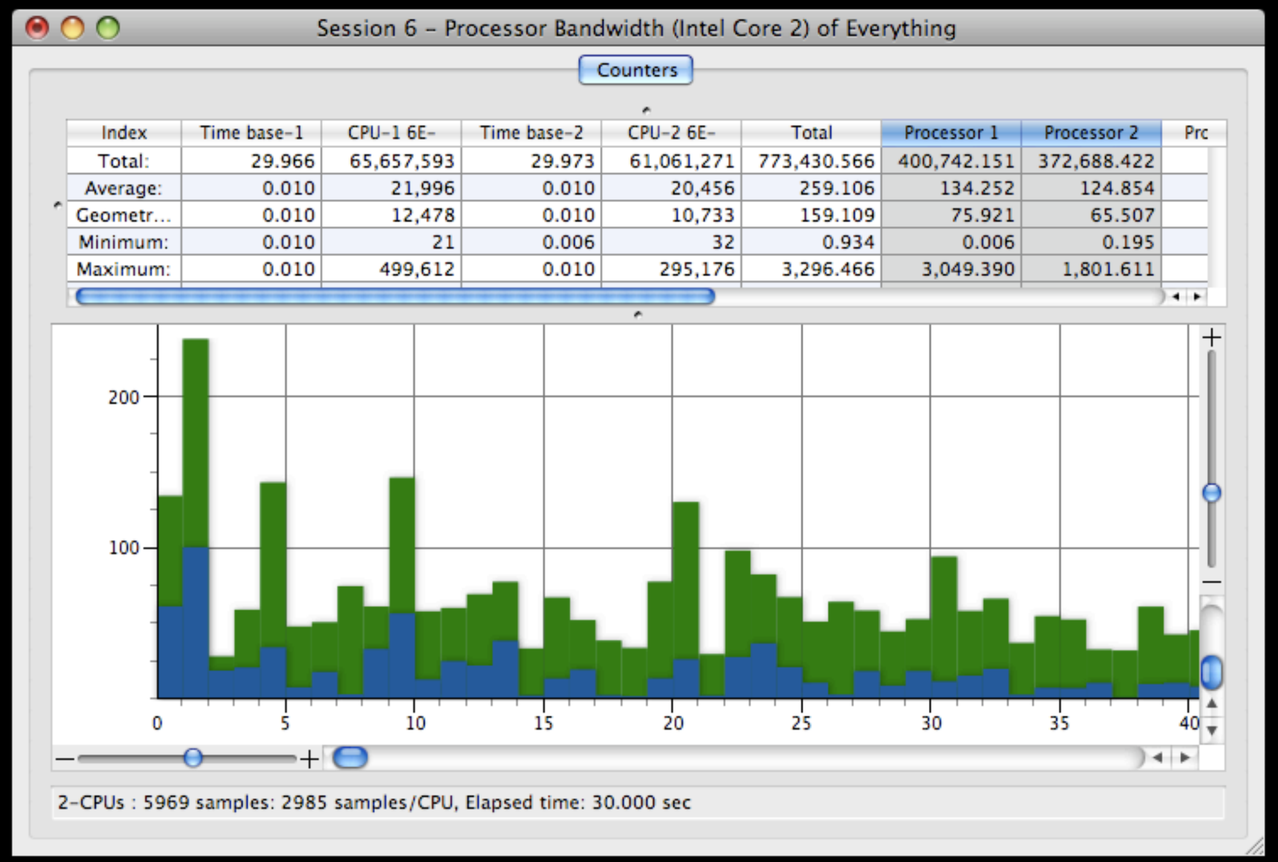


How Does Shark Help?

Processor bandwidth/L2 cache misses

Only on Mac OS

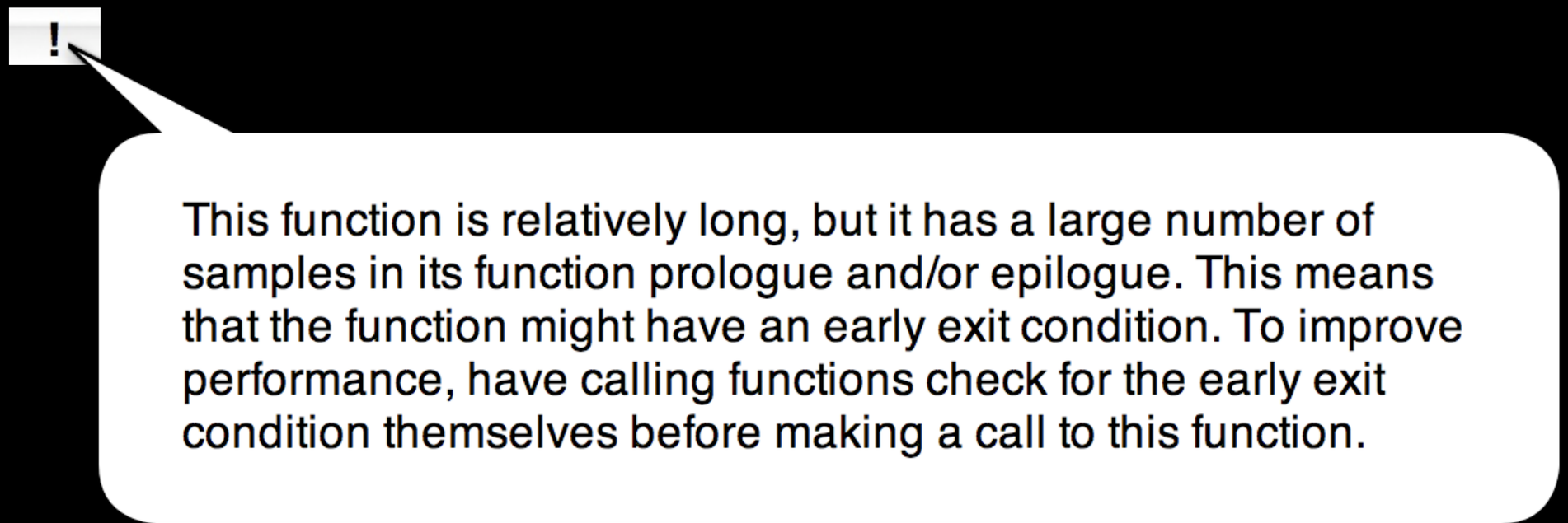
- Shows you how much bus bandwidth you're using
- Shows you where you're overflowing the CPU caches



How Does Shark Help?

Static analysis

- Flags inefficient code, down to the assembly level
- Suggests performance improvements



Getting Ready to Use Shark

- Debug your program
- Build as if you're intending to release
  - Full compiler optimization
    - Except -fomit-frame-pointers
  - Ensure your symbol files are available
- Choose your performance metric
- Take your baseline performance measurement
  - Must be measured in a repeatable manner!
  - Write a script
    - Shark can be automated

Demo

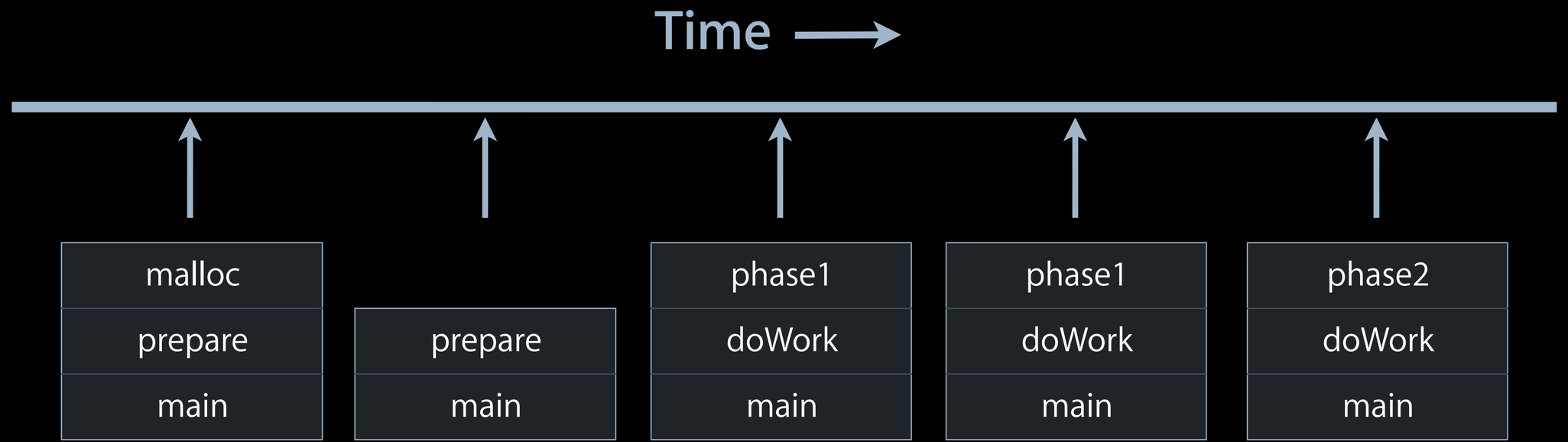
Recording a Shark session

Time Profiling

Time Profiling

How does it work?

- Shark takes a sample periodically
- Each sample contains the callstacks of the target threads
  - Including kernel time and drivers



Time Profiling

How does it work?

- A calltree is constructed by merging all these callstacks

▼ main

▼ prepare

malloc

▼ doWork

phase1

phase2

Time Profiling

How does it work?

- A calltree is constructed by merging all these callstacks
  - Including how often they appeared

Self	Total	Symbol
0%	100%	▼ main
20%	40%	▼ prepare
20%	20%	malloc
0%	60%	▼ doWork
40%	40%	phase1
20%	20%	phase2

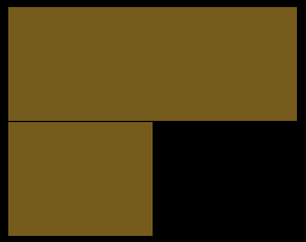
Time Profiling

How does it work?

- Self includes only the function itself, not its callers or subfunctions
- Total includes the function and any subfunctions

```
void main(void) {
    work_t *work = prepare(100);

    doWork(work);
}

work_t* prepare(unsigned int units) {
    work_t *result = malloc(sizeof(work_t) * units);

    for (int i = 0; i < units; ++i) {
        result[i] = NULL;
    }

    return result;
}
```

Self	Total	Symbol
0%	100%	▼ main
20%	40%	▼ prepare
20%	20%	malloc
0%	60%	▼ doWork
40%	40%	phase1
20%	20%	phase2



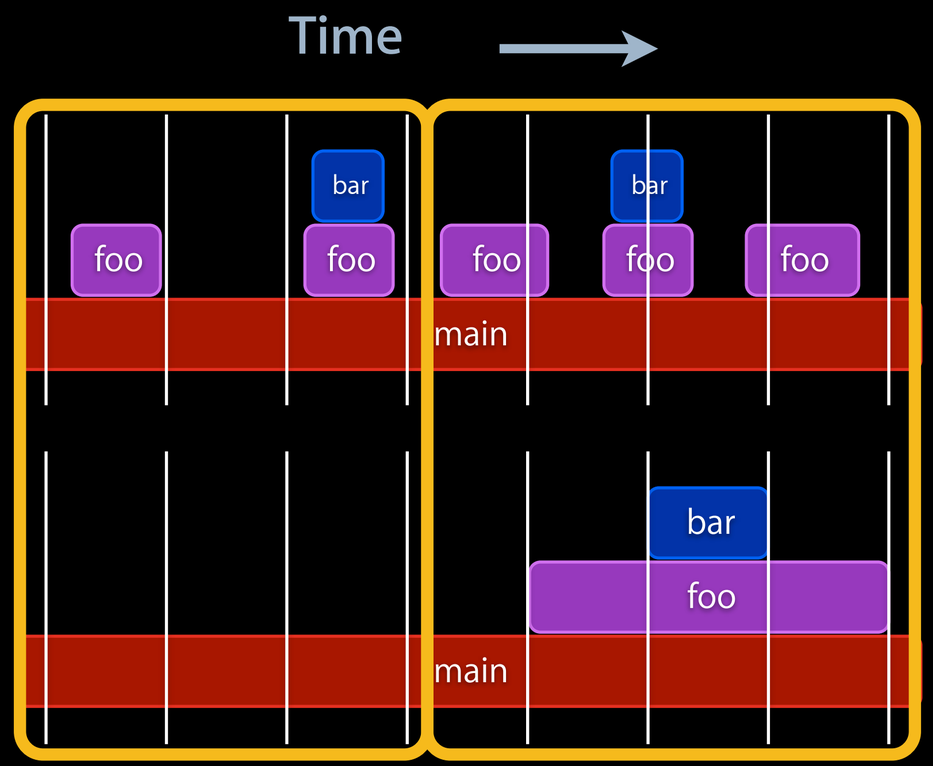
Time Profiling

Caveats

- Not a trace
  - Missing functions: Samples between small functions
  - Oversize functions: Samples right on small functions
  - Doesn't tell you how often you called a function
- Only shows you where, not why

Original Execution:

Perceived Execution:



Time Profiling

Rules of thumb

- Minimize time required to do a unit of work
- Minimize blocking for throughput
- Prioritize issues based on their relative impact vs. effort to fix

Demo

Analysis of Time Profiles

System Tracing

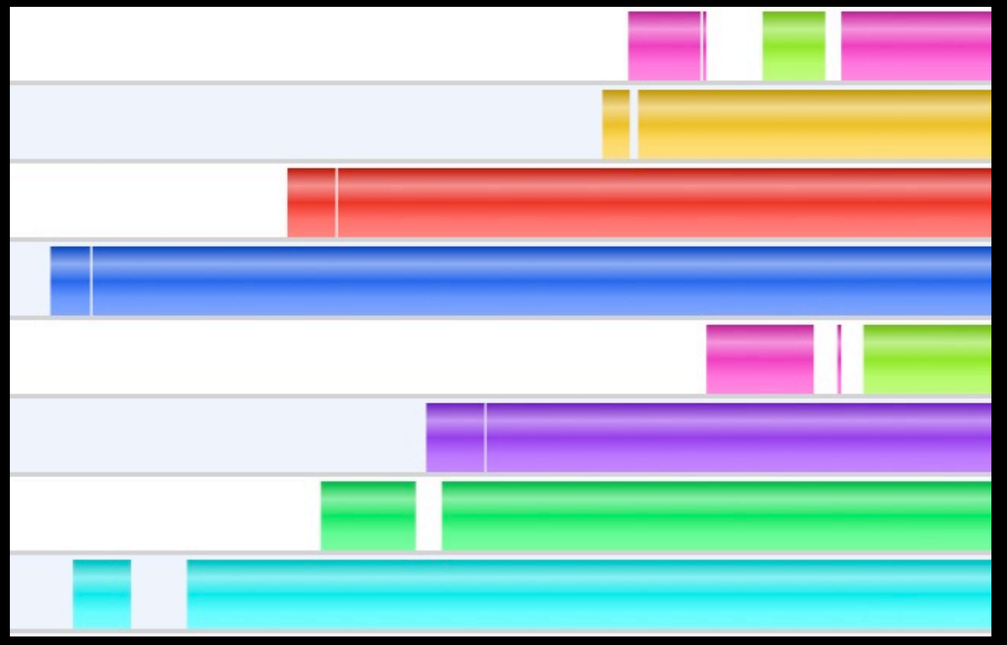
Eric Miller
Minion, Developer Technologies

System Tracing

Thread Scheduling

Only on Mac OS

- Which threads were running, when, and on what cores
- Thread priorities
- Context switch reasons

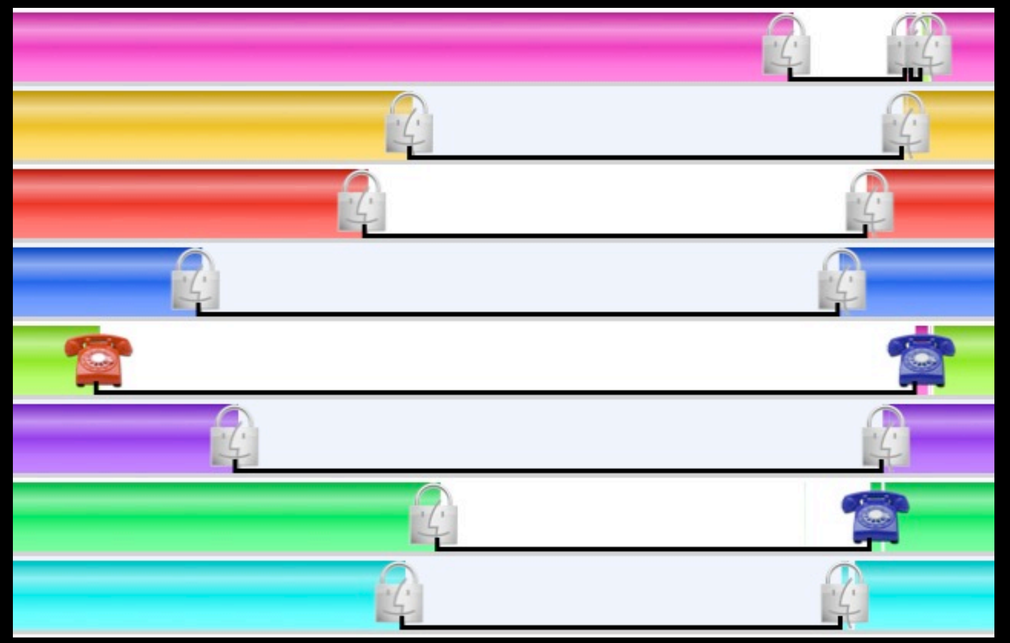


- Rules of thumb:
  - Minimize context switches
  - Maximize length of tenures for throughput
  - Minimize length of tenures for responsiveness

System Tracing

System Calls

Only on Mac OS



- Requests to the kernel
  - read, select, Mach messages, etc.
  - Underlie locks
- Incur context switch
- Can block
- Can be serializing

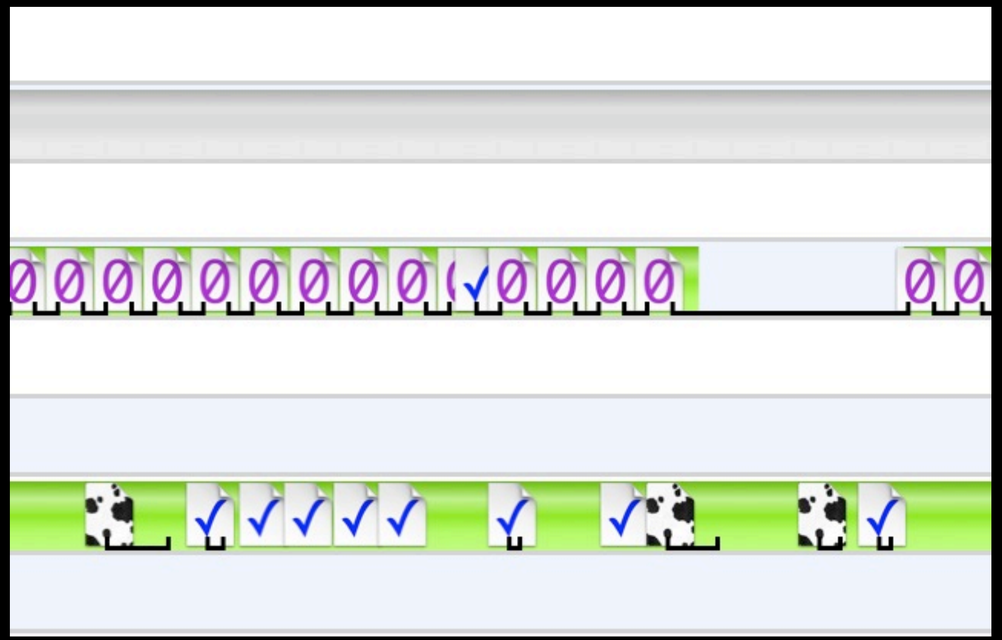
- Rules of thumb:
  - Minimize system calls
  - Avoid slow and blocking calls for throughput

System Tracing

Virtual Memory Faults

Only on Mac OS

- Use of an unmapped page
  - First use of new allocations
  - First write to a shared page
  - Page-ins (and outs) from disk



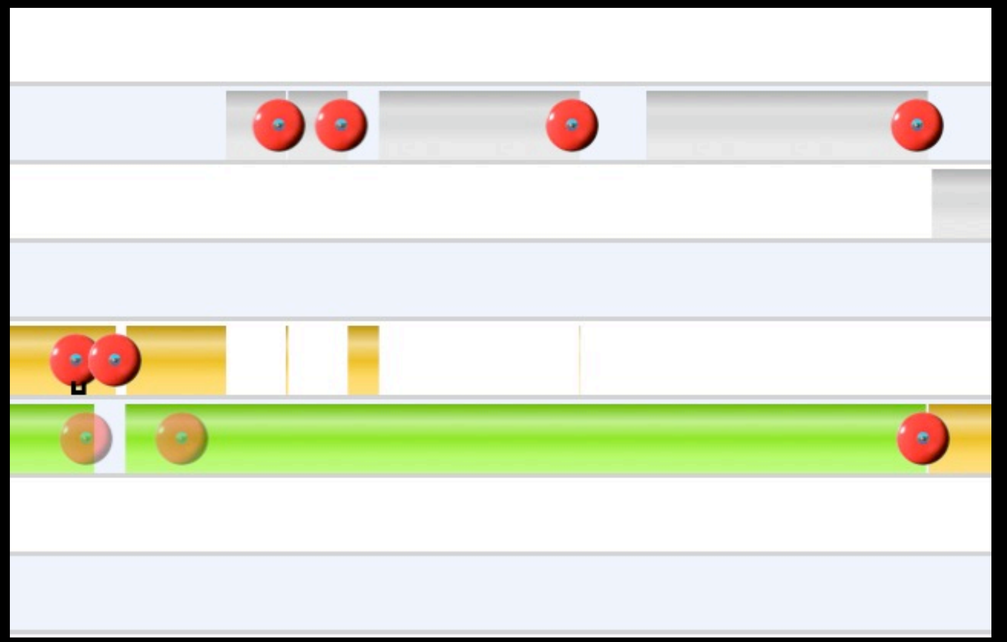
- Rules of thumb:
  - Minimizing memory use generally minimizes page faults
  - Reuse memory allocations
  - Preflight or wire pages for latency-sensitive applications

System Tracing

Interrupts

Only on Mac OS

- Asynchronous signals to the CPU
  - Network and disk hardware
  - PCI devices
  - Hardware timers



- Rules of thumb:
  - If you don’t know about them, you probably don’t need to
  - Minimize interrupts if you’re a source of them (e.g. a driver)

Demo

System Trace

System Tracing

Signposts in user space

Only on Mac OS

```
#include <sys/kdebug.h>
#include <sys/syscall.h>

work_t* prepare(unsigned int units) {
    syscall(SYS_kdebug_trace, APPSDBG_CODE(DBG_MACH_CHUD, kPrepare) | DBG_FUNC_START,
            units, 0, 0, 0);

    work_t *result = malloc(sizeof(work_t) * units);

    for (int i = 0; i < units; ++i) {
        syscall(SYS_kdebug_trace, APPSDBG_CODE(DBG_MACH_CHUD, kPrepareUnit),
                units, i, result, &result[i]);
        result[i] = NULL;
    }

    syscall(SYS_kdebug_trace, APPSDBG_CODE(DBG_MACH_CHUD, kPrepare) | DBG_FUNC_END,
            units, 0, result, 0);

    return result;
}
```

System Tracing

Signposts in kexts

Only on Mac OS

```
#include <sys/kdebug.h>


work_t* prepare(unsigned int units) {
    KERNEL_DEBUG_CONSTANT(APPSDBG_CODE(DBG_MACH_CHUD, kPrepare) | DBG_FUNC_START,
            units, 0, 0, 0, 0); // 5th user argument unused
    work_t *result = malloc(sizeof(work_t) * units);

    for (int i = 0; i < units; ++i) {
        KERNEL_DEBUG_CONSTANT(APPSDBG_CODE(DBG_MACH_CHUD, kPrepareUnit),
                units, i, result, &result[i], 0); // 5th user argument unused
        result[i] = NULL;
    }

    KERNEL_DEBUG_CONSTANT(APPSDBG_CODE(DBG_MACH_CHUD, kPrepare) | DBG_FUNC_END,
            units, 0, result, 0, 0); // 5th user argument unused

    return result;
}
```

System Tracing

Signposts in DTrace

Only on Mac OS

```
enum signposts {
    kFunctionCall = 0x10,
};

pid$1::$2:entry {
    core_profile(COREPROFILE_SIGNPOST_START, kFunctionCall, arg0, arg1, arg2, arg3);
}

pid$1::$2:return {
    core_profile(COREPROFILE_SIGNPOST_END, kFunctionCall, arg0); // Up to four user args
}
```

System Tracing

Signpost names

Only on Mac OS

- Create a signpost names file for Shark in:

```
~/Library/Application Support/Shark/KDebugCodes/
```

```
0x31 kPrepare
0x32 kPrepareUnit
0x40 kWork
0x41 kWorkUnit
...
```

Bandwidth and L2 Cache Misses

Processor Bandwidth Profiling

Your computer is a series of tubes

Only on Mac OS

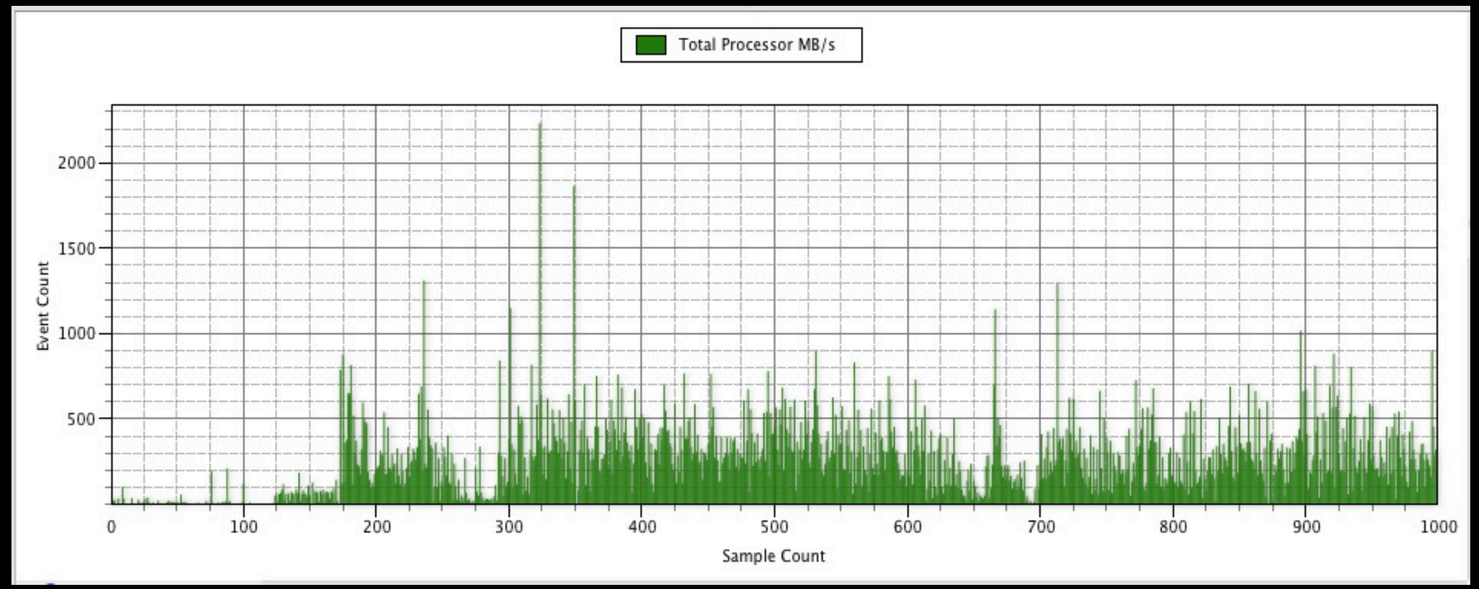
- Shows how much processor bandwidth is being used
  - Memory bandwidth is typically less than processor bandwidth
- Maximum bandwidth varies by machine and type of workload
  - Theoretical bandwidth is never achieved in practice

- Rules of thumb:
  - Intel machines typically capable of ~90% of theoretical bandwidth
  - “Effective” memory bandwidth can be much lower than actual

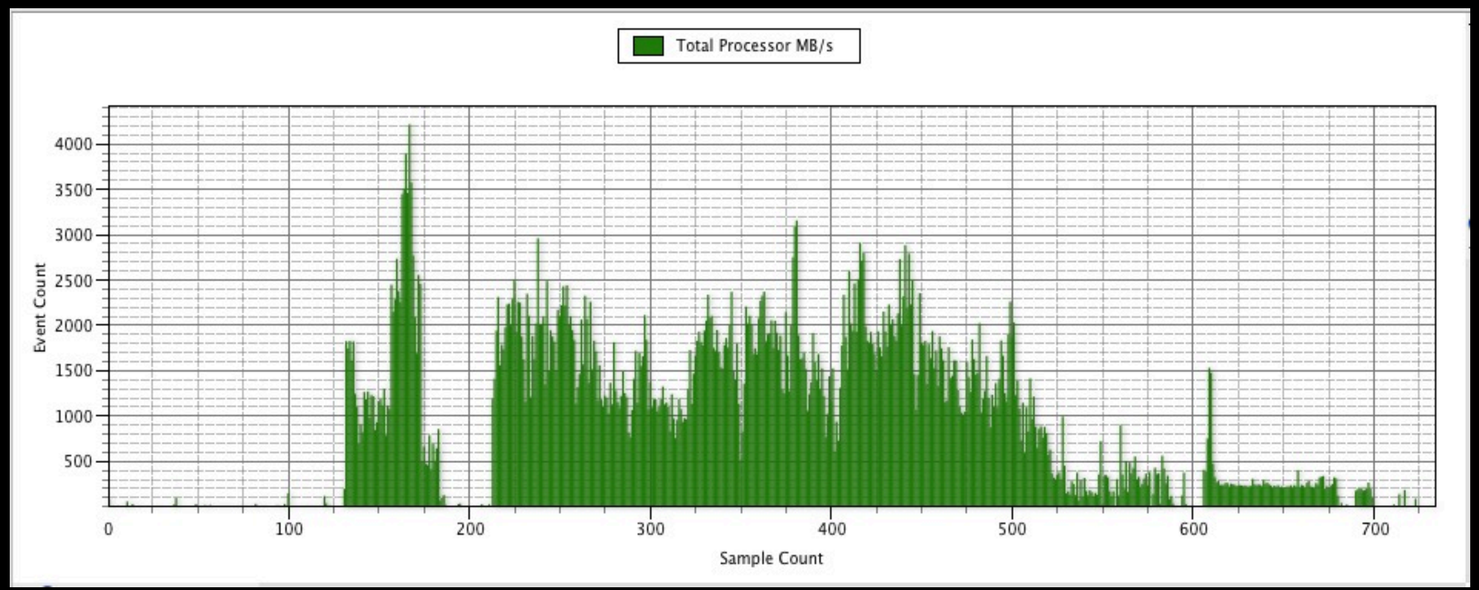
Processor Bandwidth Profiling

Before and after…

ImageAware Single Thread



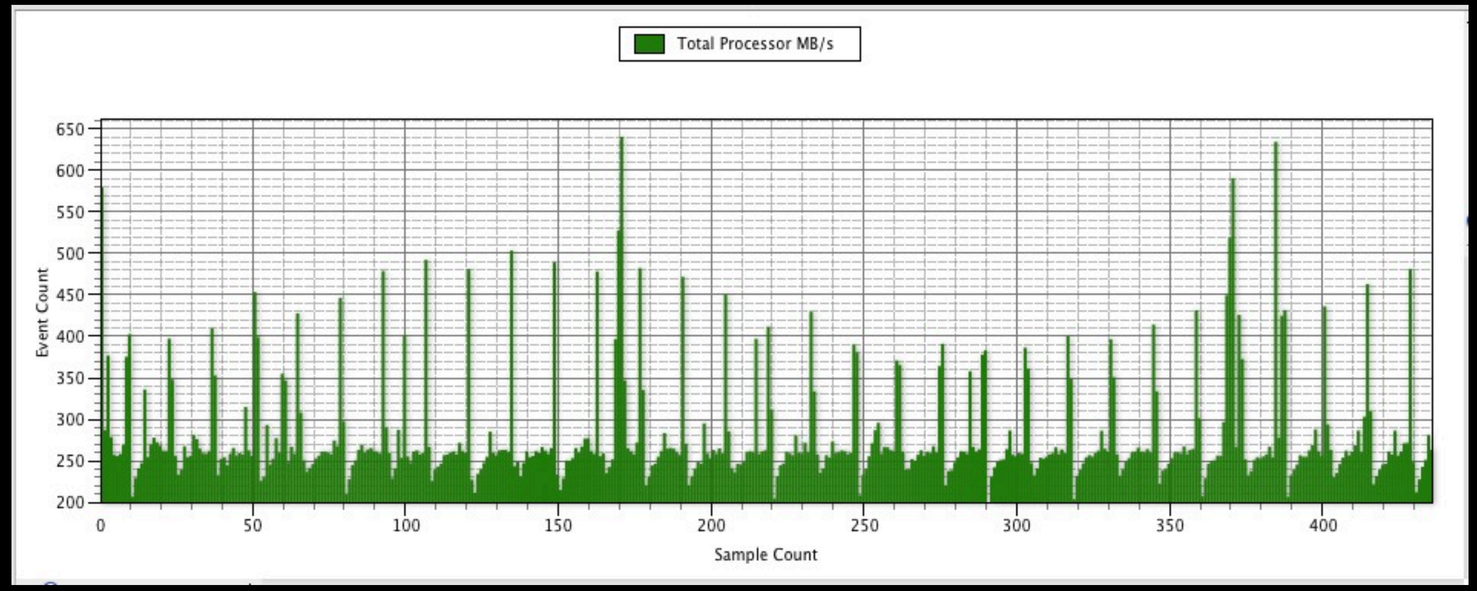
ImageAware GCD



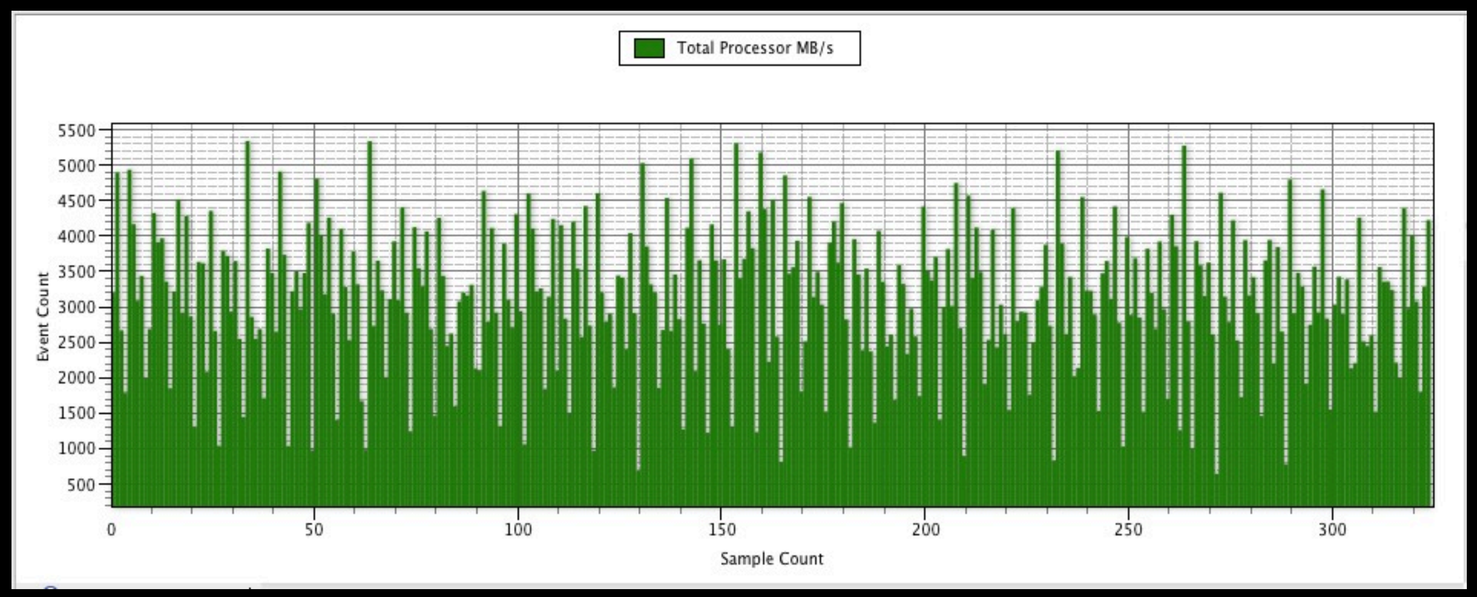
Processor Bandwidth Profiling

Before and after…

Noble Ape
Single Thread



Noble Ape
Optimized
and Threaded



L2 Cache Miss Profiling

Location, location, location

Only on Mac OS

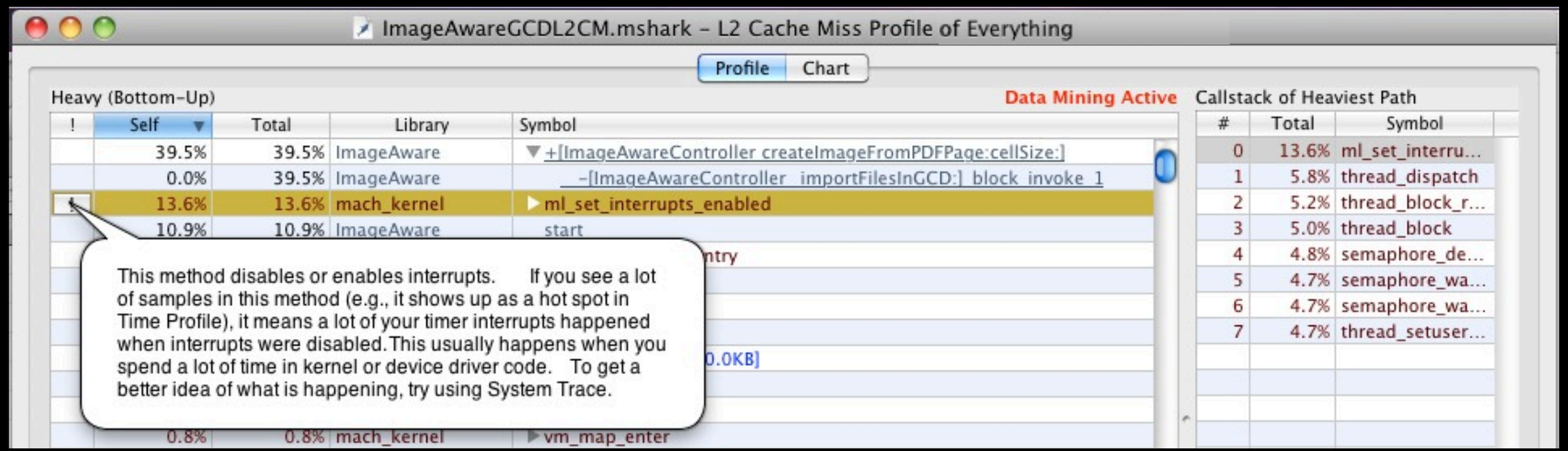
- Shows where you are missing the processor caches
  - Misses take time and stall your program
- May indicate an inefficient data layout or access pattern
  - Time Profiles don't readily reveal this

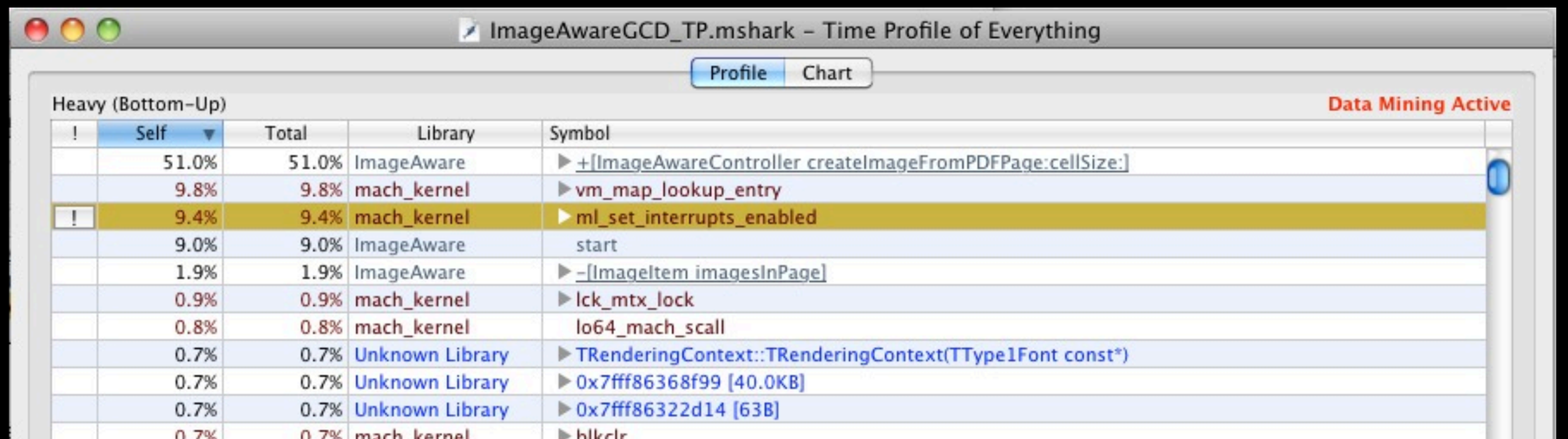
- Rules of thumb:
  - Keep your working set within the cache
    - Maximize memory access locality
  - Be wary of Garbage Collection

L2 Cache Miss Profiling

Comparison shopping…

Only on Mac OS





Summary

Wrap it up—I'll take it

- Time Profiling is very broadly useful and a good place to start
- System Trace is particularly useful for certain types of applications
  - Multithreaded
  - I/O-heavy
  - Real time (or similar)
- Bandwidth and Cache Miss profiles particularly useful for large data sets
- Shark can do so much more
  - Check out the Shark User Guide from the Help menu

More Information

Mike Jurewitz
Development Tools and Performance Evangelist
jurewitz@apple.com

Performance Tools Mailing List
perftools-feedback@group.apple.com

Documentation
Shark User Guide
http://developer.apple.com/documentation/

Related Session

Performance Tuning with Shark and Intel's Processor Performance Events	Pacific Heights Friday 10:30AM

Labs

Performance Optimization with Shark Tutorial	iPhone Lab C Friday 10:30AM
Performance Tuning with Shark on Mac and iPhone Lab	iPhone Lab C Friday 2:00PM
iPhone Performance Optimization with Instruments Lab	iPhone Lab B Friday 2:00PM

Q&A

WWDC Session Survey

You can rate your sessions on the WWDC Attendee Site.

Survey forms are located on each session description page at:

developer.apple.com/wwdc/attendee

WWDC09